Course Name : Object Oriented Analysis & Design (OOAD)

Chapter 00: Analysis and Design with UML

- Unified Modeling Language (UML)
- Why Construct Model?
- UML Diagram
- Type of UML Diagram

Chapter 01: Object Oriented Analysis and Design (OOAD)

- Object Orientation (OO) and Object Oriented Method (OOM)
- Object Oriented Modeling
- Polymorphism
- Inheritance
- Object Oriented Analysis and Design
- Jacobson Use Case Method (OO Software Engineering – OOSE)

- System Development
  - Requirement Model
  - Analysis Model
  - Design Model
  - Implementation Model
  - Test Model
  - Unified Modeling Language (UML)
    - The Value of UML
    - History of UML
    - Overview of the UML

Chapter 02: Complexity

- Complexity
- Complexity Crisis
- General Idea of How to Deal with Complexity
- The Structure of Complex System
- Five Attributes of a Complex System
- Bringing Order to Chaos
- On Designing Complex Systems

Chapter 03: The Object Model (OM)

- Element of Object Model
  - Four Major Elements of OM
  - Three Minor Elements of OM

Chapter 04: Classes and Objects

- Nature of an Objects
- Relationship among Objects
- Nature of Class
- Relationship among Classes

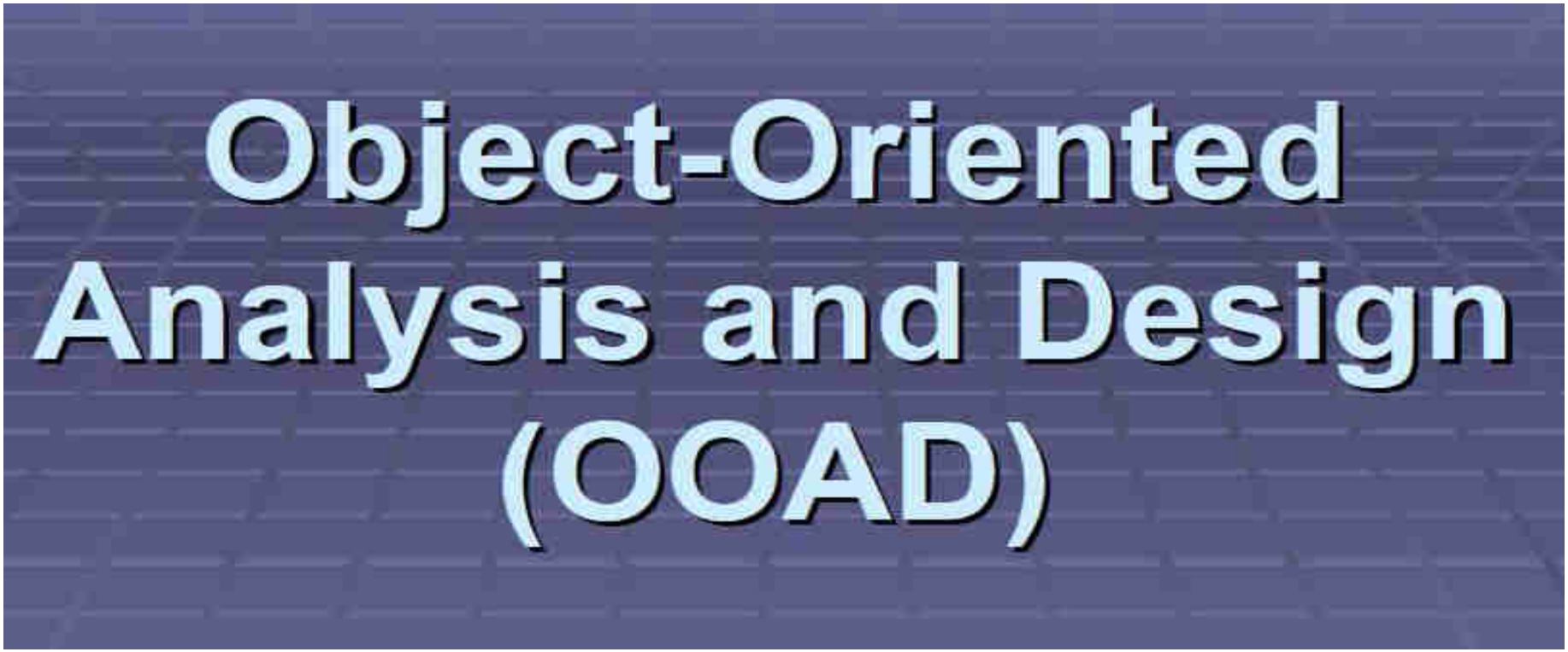
Chapter 05: Basic OOAD Process

- Basic OO Concept
- Basic OOAD Process
- The four Ps

Chapter 06: Project Management

- Project Management
- Software Project Management
- Project Plan Structure
- The Risk Management Process
- Organization of the Software Project Management Plan Document

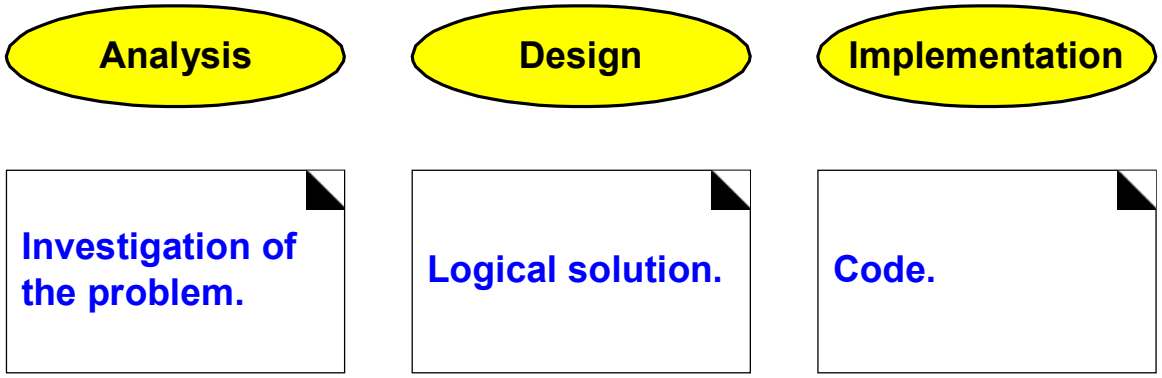
Chapter 01



What Is an Object?

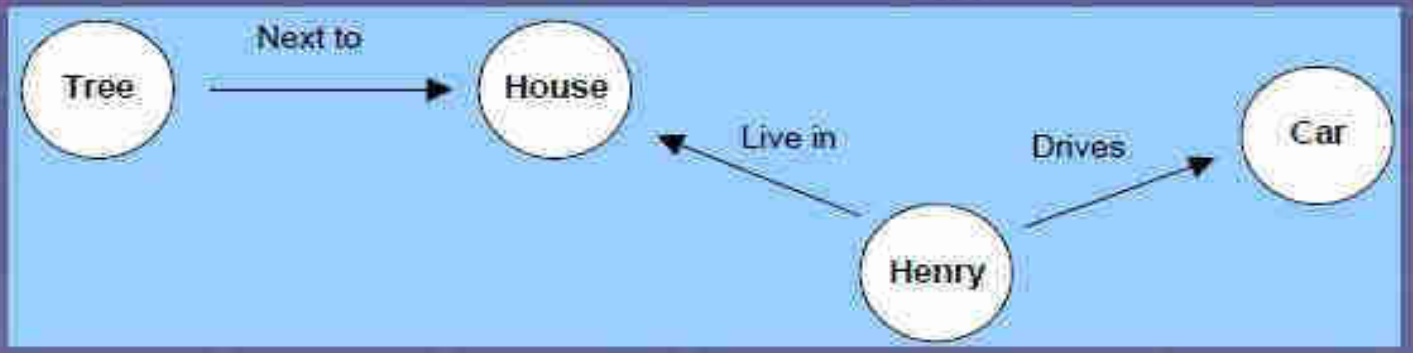
- **Definition:** An object is a software bundle of variables and related methods.
- As the name *object-oriented* implies, *objects* are key to understanding object-oriented technology.
- You can look around you now and see many examples of real-world objects: your dog, your desk, your television set, your bicycle.
- Software objects are modeled after real-world objects in that they, too, have state and behavior.
- A software object maintains its state in *variables* and implements its behavior with *methods*.

Object-Orientation



- Object-oriented analysis.
  - Investigation that is object-centric.
- Object-oriented design.
  - Solution in terms of interacting software objects.
- Object-oriented programming.
  - Coding in an object-oriented programming language.

OO Modelling



The basics

- Objects
- Classes
- Relationships
- An Instance
- Idea of encapsulation

An Object

- Some concept of reality
- A physical entity
- It is characterised by:
  - a number of operations,
  - a state which remembers the effect of these operations

An object

- **Operations:**
  - Work
  - Dance
  - Drive
  - Jump
- **Attributes:**
  - Height
  - Eye colour
  - Hair colour
  - Weight

Relationships

- Static:
  - relations existing over a long time
  - objects know about each other existence
- Dynamic:
  - relations which two objects communicate with each other
  - object sending stimuli to other
  - stimuli - events, messages

'Creating' objects

- **Composition** - structure object from parts
- **Partition** - into hierarchy ('**is a**')
- **Consist of** - build objects from others
- **Aggregate** - to join together ('**has a**')

Encapsulation

- A concept of **'Self-containing'**
- **Information hiding** - 'internal' structure is hidden from their surroundings
- Behaviour and information is represented or implemented **internally**
- Functionality and behaviour characterised by '**interfacing**' operations

Class

- A **class** represents a *template* for several objects and describes how these objects are *structured internally*
- Objects of the same class have the same definition both for their operations and their information structure
- Class is an **implementation** of objects

Key Concepts

- Polymorphism – same object has different implementations
- Inheritance – to adopt, permutate, and derive from some generic objects

Polymorphism

- A concept in type theory
- A **common** name may denote instances of different classes
- One type of operation can be implemented in different ways by different classes
- ***Overloading*** in modern OO language

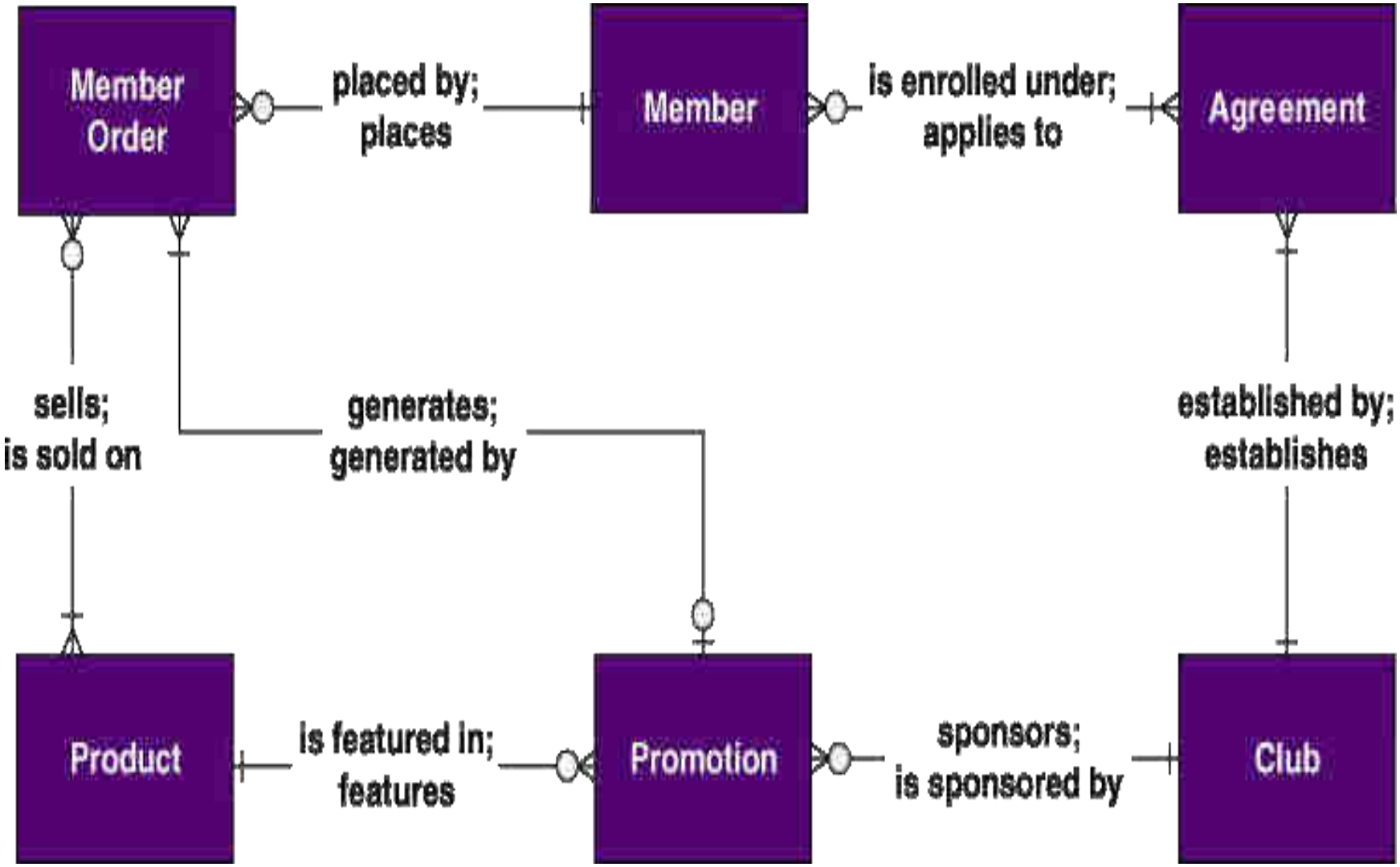
Why Polymorphism

- A very strong tool for allowing system designers to develop *flexible* systems
- Designer only need to specify *what* shall occur and not *how* it shall occur
- To add an object, the modification will only affect the new object, not those using it

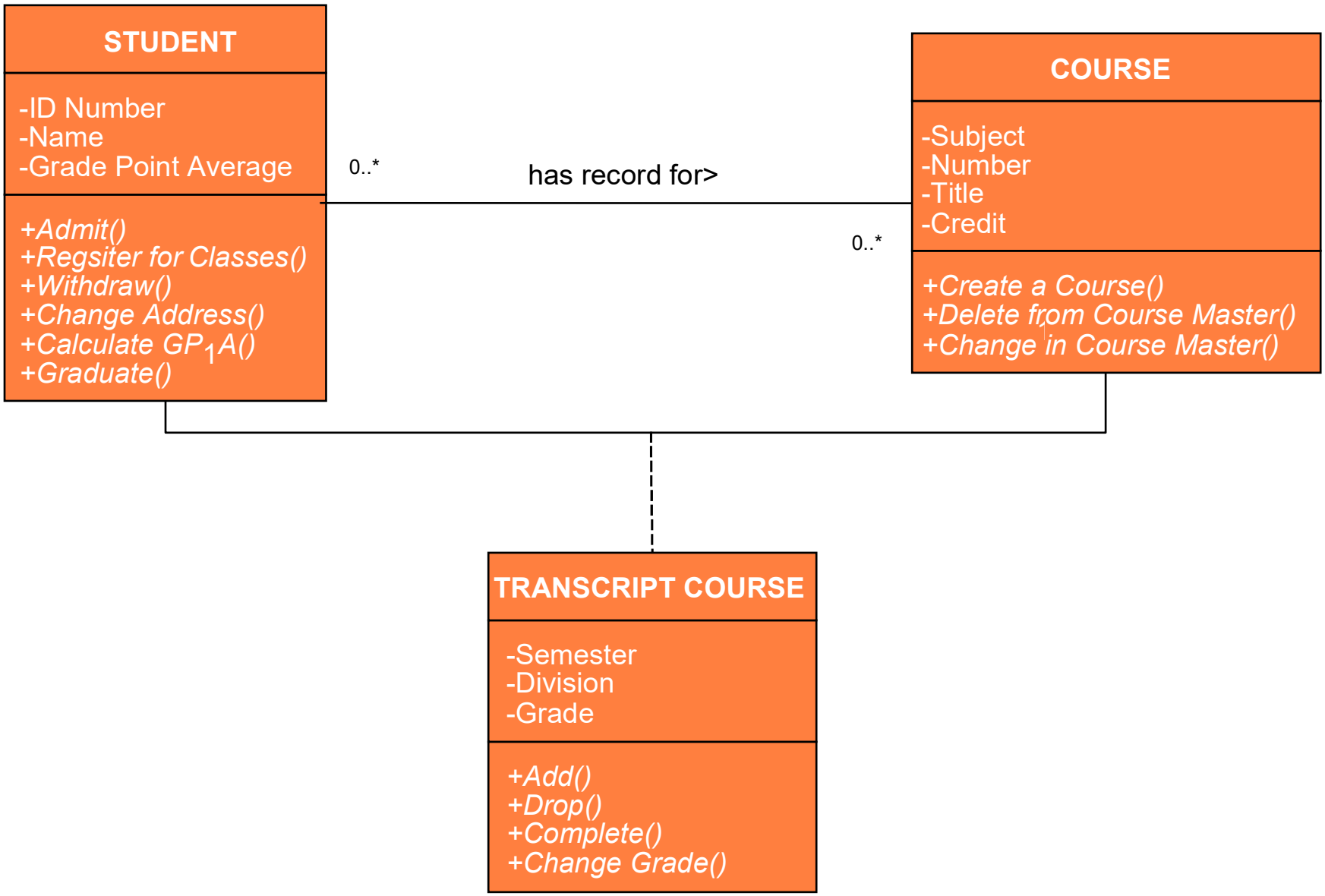
Inheritance

"If class B inherits class A, then both operations and the information structure described in class A will become part of class B"

A Simple Data Model



A Simple Object Model



Object-Oriented Methods

- Advocate integral ***objects*** which encapsulate both function and data
- Main activities include:
  - Identification of objects, and
  - Analysing their behaviour and information
- Uses object-oriented techniques and ideas:
  - inheritance
  - polymorphism
  - function/data abstraction

Object-Oriented Analysis & Design

1. Finding objects
2. Organising objects
3. Describing how objects interacts
4. Defining the operations of objects
5. Defining objects internally

1. Finding Object

Finding Objects

- Naturally occurring entities - physical
- A concept of some abstract ideas - conceptual
- Should be *stable*
- Classes of objects
  - active/passive
  - temporary/permanent/persistent
  - part/whole
  - generic/specific
  - private/public

2. Organizing Object

Organising Objects

- Classification
- Similar objects - inheritance: '**is a**'
- Interactions/relationships between objects
- Whole/Part relationship: '**has a**'

3. Describing how objects interacts

Object Interactions

- Identify ***how*** objects fit into a system
- Use of ***scenarios*** - unique situations
- Objects' ***communication***
- Objects' ***interfaces***
- Refined relationships

4. Defining the operations of objects

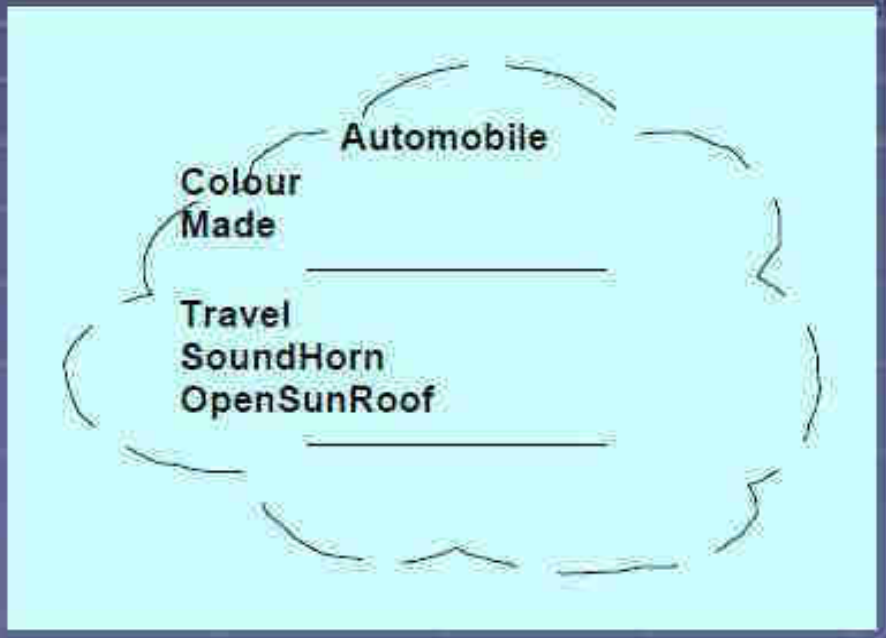
Object Functionality, Object Implementation & Testing of Objects

Object's Functionality

- Operations performed by an object
- Behaviour of an object
- Specification of interfaces, external and internal functions
- Objects with complex functionality should be partitioned into simpler objects

Object Implementation

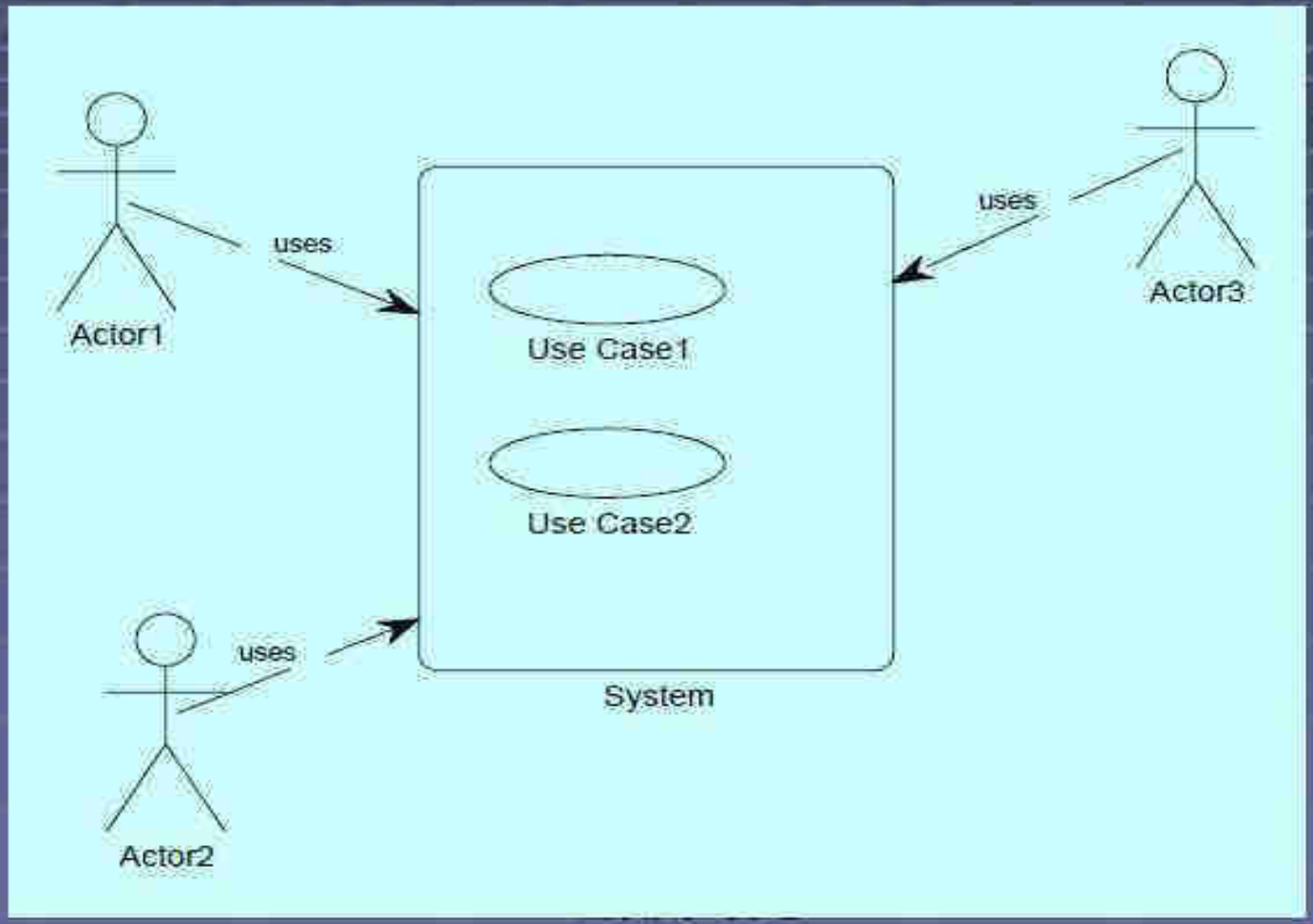
- The specification of **CLASSES**
- Define information that an object encapsulates - **ATTRIBUTES** and **METHODS**



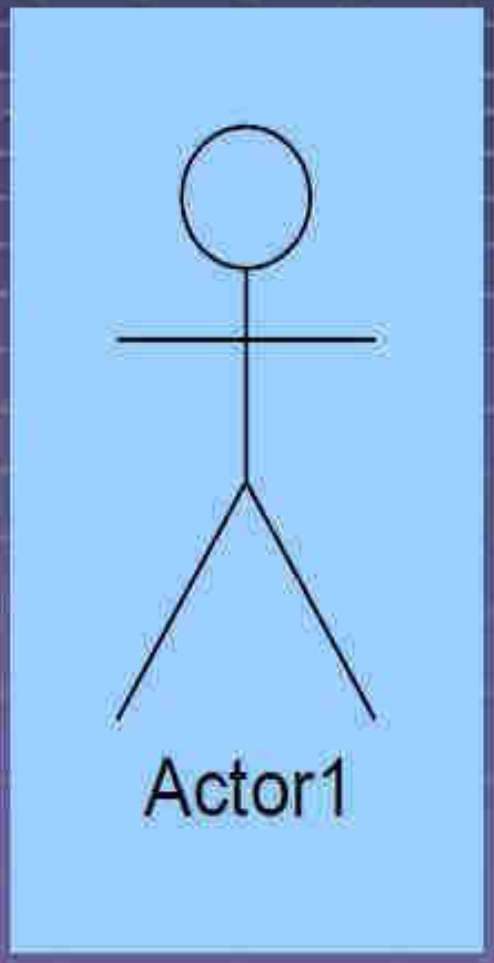
Object Implementation

- **METHODS**:
  - Specify external functions
  - Specify internal functions that are not seen or usable by others objects
- **Languages**: C++, Smalltalk, Ada, Eiffel, Modula-2, Simula, Java++

Requirement - Use Case Model

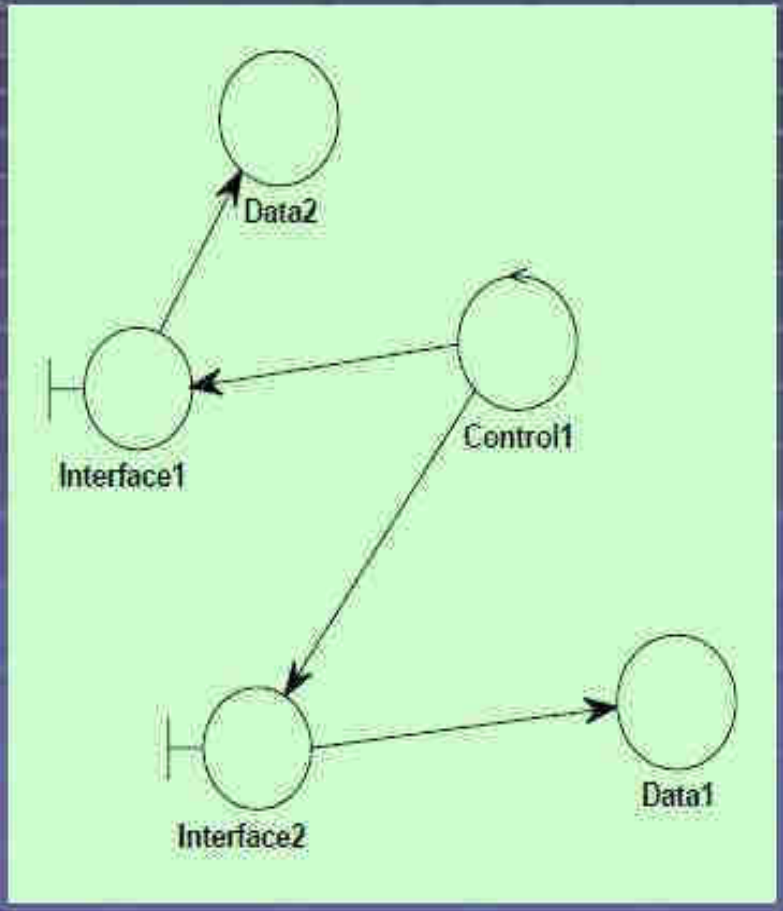


Requirement Model



- Actors:
  - Essential system entities from an user view point
  - Interacts with system
  - Changes system behavior
  - Control system functionality

Analysis Model



- **Structure** a system independently to the actual implementation
- Capture information, behavior and presentation
- Specify objects

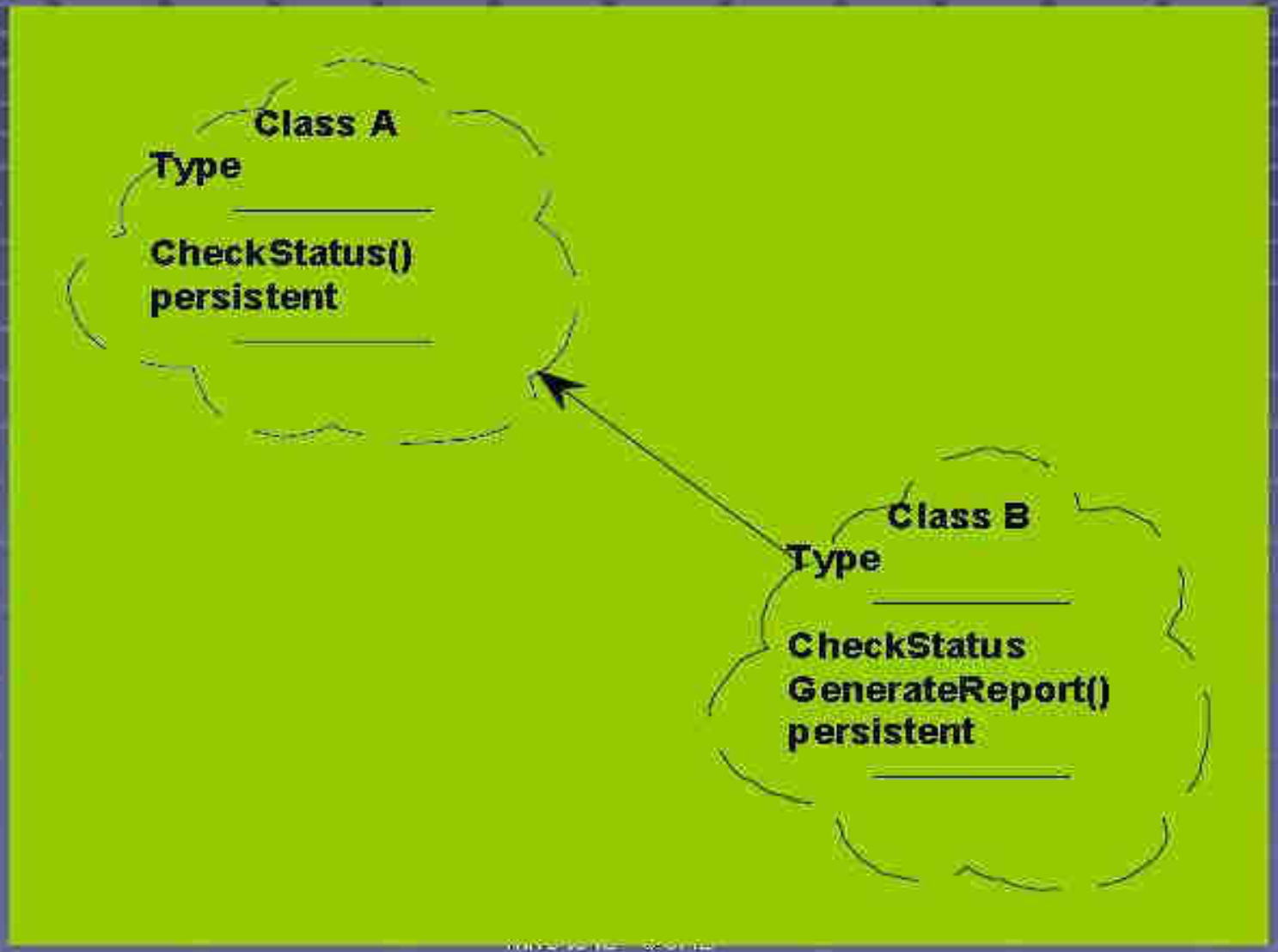
Design Model

- Refine the object structure to the chosen implementation environment
- Objects are consolidated into '**blocks**' - abstracted classes
- Blocks interactions are also documented using interaction diagrams

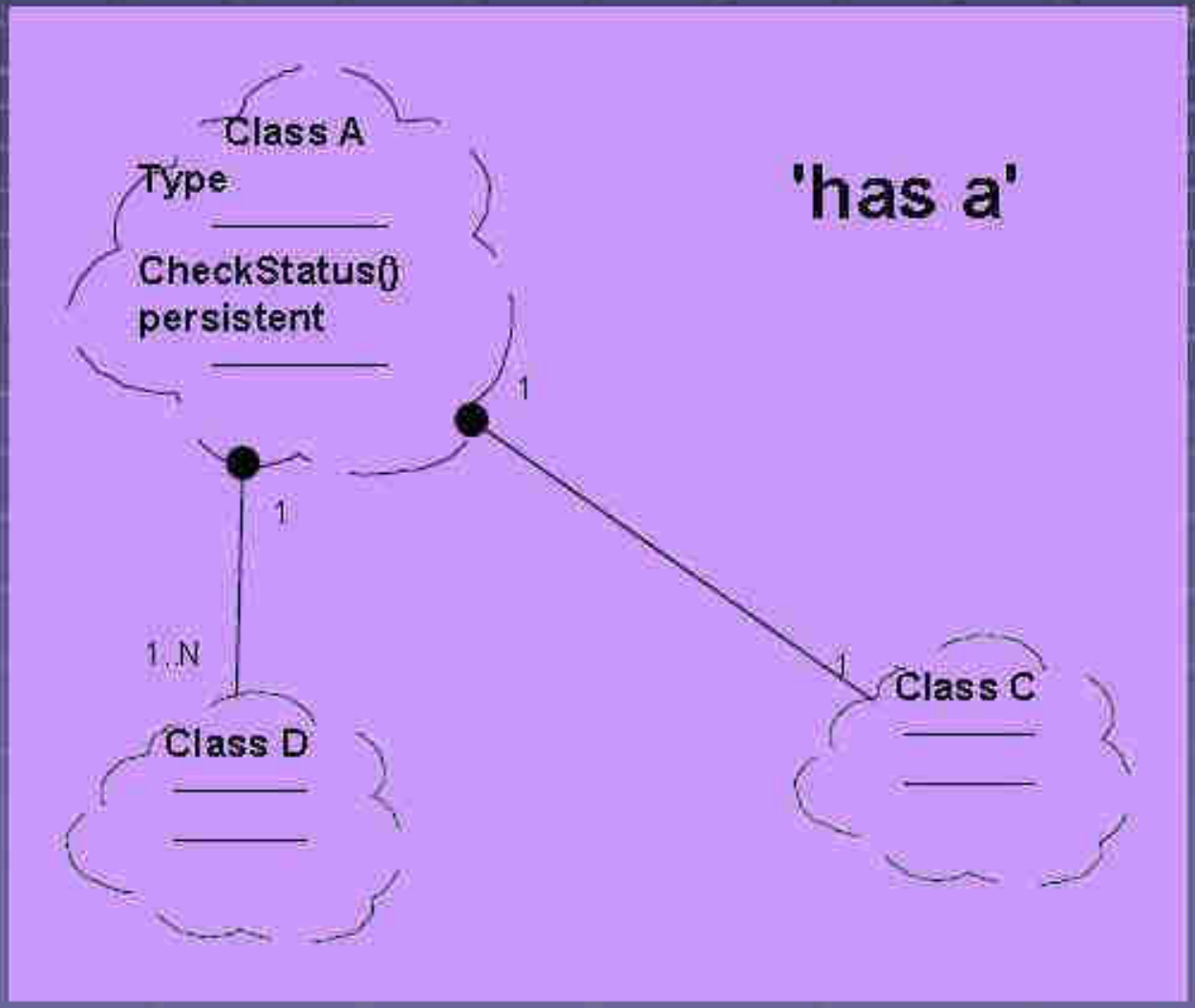
Implementation Model

- The blocks in the design model are implemented using classes
- Class diagrams are used to express relationships between classes
- Class specifications
- Annotated source code (pseudo-code) for methods and attributes of classes

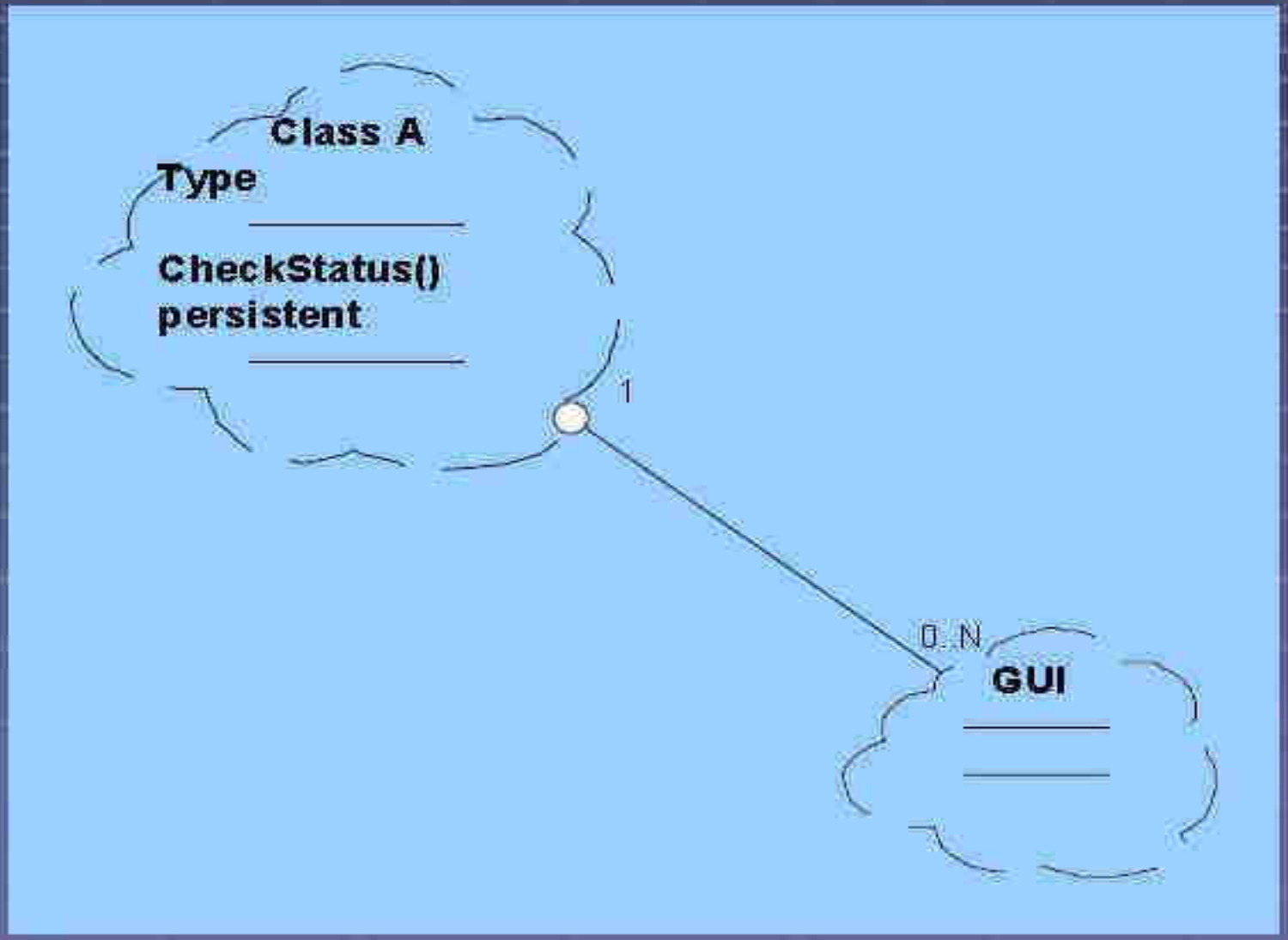
Class Diagrams - Inheritance



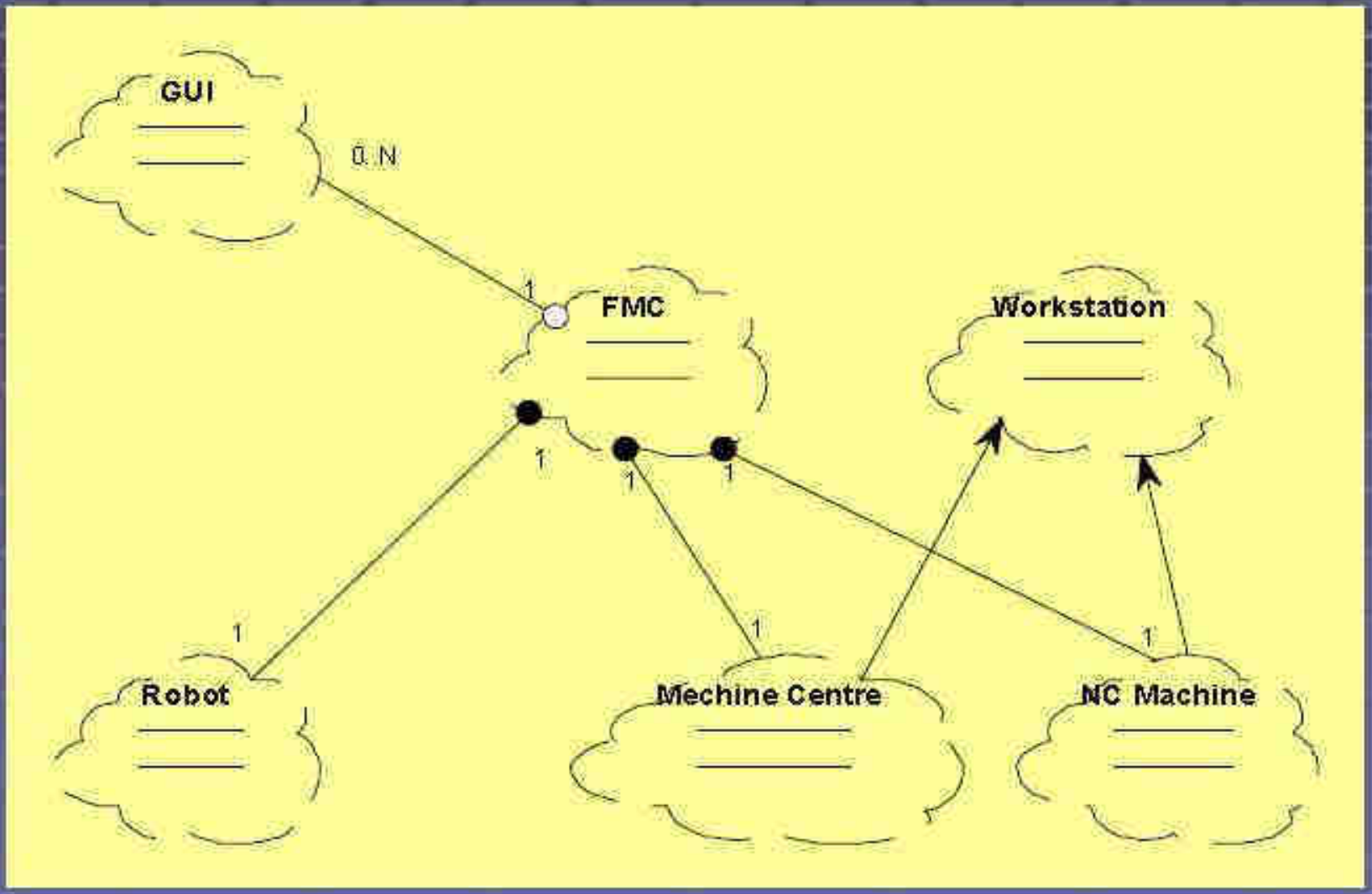
Class Diagram - Aggregation



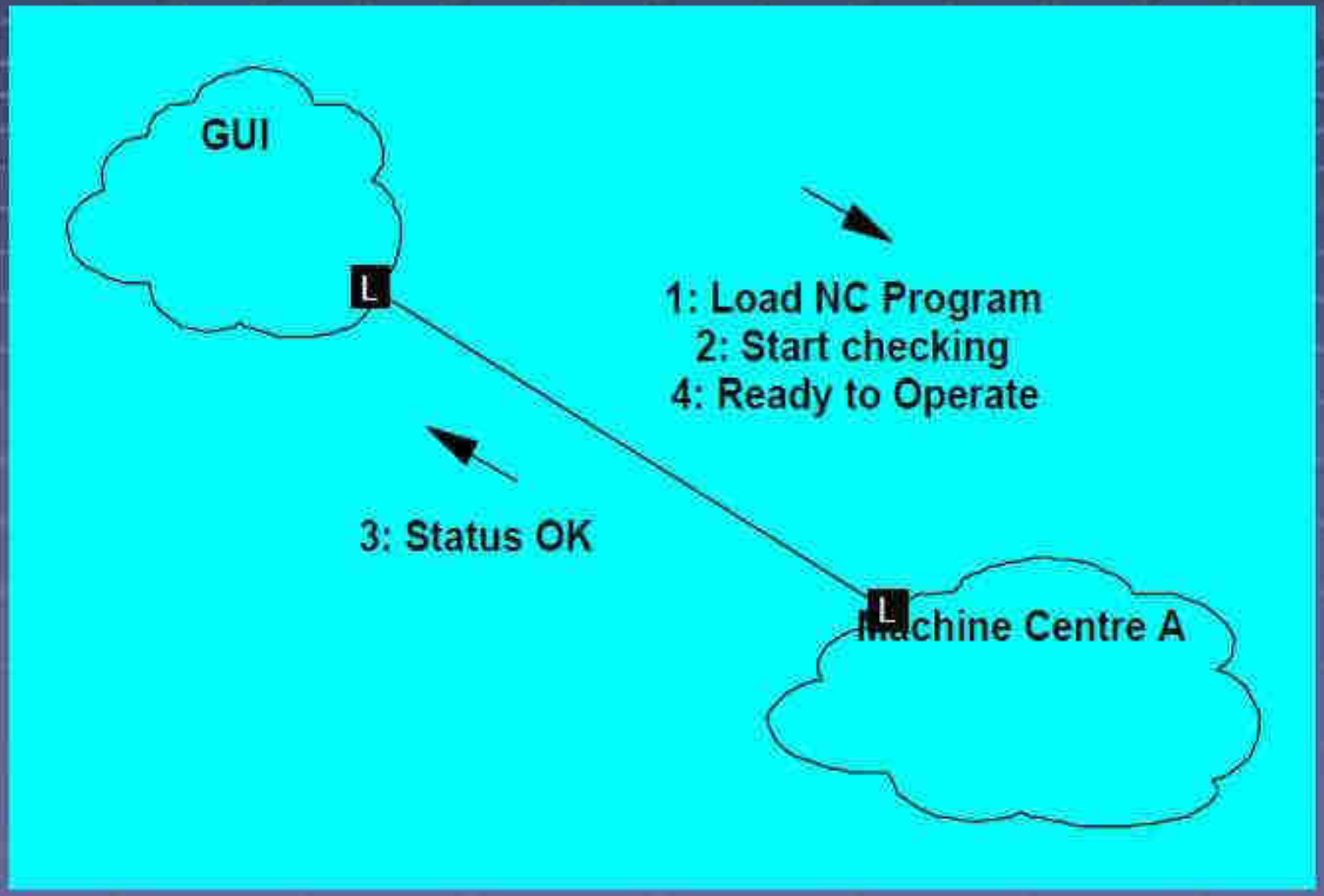
Class Diagram - 'Uses'



Class Diagram - An Example



Object Diagrams



Test Model

- States the method and result of testing
- Test specification describe how classes and system are to be tested
- Test results document outcome of the tests executed
- Verification and validation

Unified Modeling Language (UML)

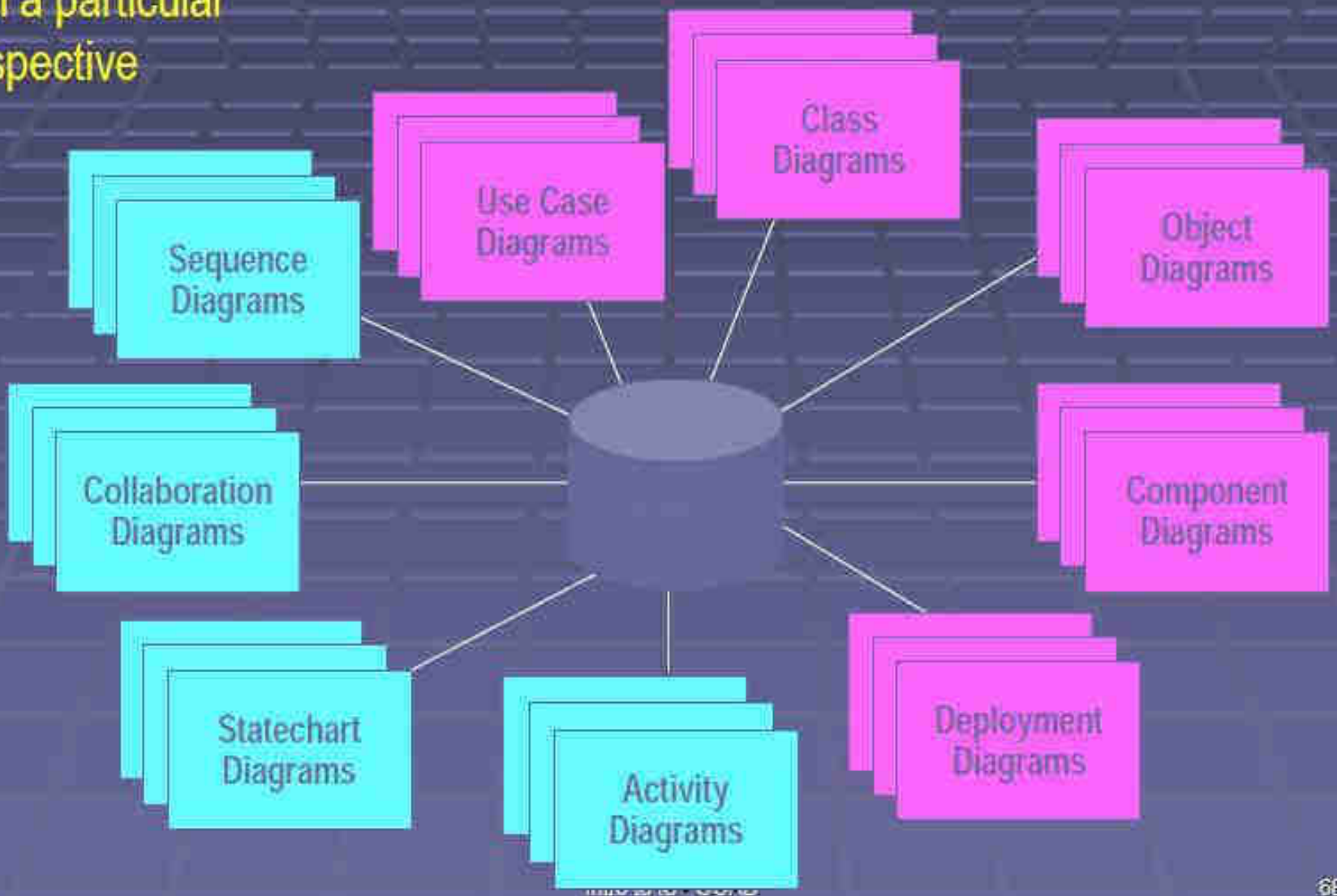
Overview of the UML

- The UML is a language for
  - visualizing
  - specifying
  - constructing
  - documenting

the artifacts of a software-intensive system

Graphical Tools

A ***model*** is a complete description of a system from a particular perspective



The End